Using the Communications Control

See Also

The Communications control allows you to add both simple serial port communication functionality to your application and advanced functionality to create a full-featured, event-driven communications tool.

The Communications Control

The Communications control provides an interface to a standard set of communications commands. It allows you to establish a connection to a serial port, connect to another communication device (a modem, for instance), issue commands, exchange data, and monitor and respond to various events and errors that may be encountered during a serial connection.

Possible Uses

- To dial a phone number.
- To monitor a serial port for incoming data.
- To create a full-featured terminal program.

Sample Applications: Dialer.vbp and VBTerm.vbp

The Dialer.vbp and VBTerm.vbp sample applications which are listed in the samples directory, demonstrate simple and complex (respectively) programming techniques of the Communications control.

Basics of Serial Communications

Every computer comes with one or more serial ports. They are named successively: COM1, COM2, and so on. On a standard PC, the mouse is usually connected to the COM1 port. A modem may be connected to COM2, a scanner to COM3, etc. Serial ports provide a channel for the transmission of data from these external serial devices.

The essential function of the serial port is to act as an interpreter between the CPU and the serial device. As data is sent through the serial port from the CPU, Byte values are converted to serial bits. When data is received, serial bits are converted to Byte values.

A further layer of interpretation is needed to complete the transmission of data. On the operating system side, Windows uses a communications driver, Comm.drv, to send and receive data using standard Windows API functions. The serial device manufacturer provides a driver that connects its hardware to Windows. When you use the Communications control, you are issuing API functions, which are then interpreted by Comm.drv and passed to the device driver.

As a programmer, you need only concern yourself with the Windows side of this interaction. As a Visual Basic programmer, you need only concern yourself with the interface that the Communications control provides to API functions of the Windows communications driver. In other words, you set and monitor properties and events of the Communications control.

Establishing the Serial Connection

The first step in using the Communications control is establishing the connection to the serial port. The following table lists the properties that are used to establish the serial connection:

Properties	Description
CommPort	Sets and returns the communications port number.
Settings	Sets and returns the baud rate, parity, data bits, and stop bits as a string.
PortOpen	Sets and returns the state of a communications port. Also opens and closes a port.

Opening the Serial Port

To open a serial port, use the CommPort, PortOpen, and Settings properties. For example:

```
' Open the serial port
MSComm1.CommPort = 2
MSComm1.Settings = "9600,N,8,1"
MSComm1.PortOpen = True
```

The CommPort property sets which serial port to open. Assuming that a modem is connected to COM2, the above example sets the value to 2 (COM2) and

connects to the modem. You can set the CommPort property value to any number between 1 and 16 (the default is 1). If, however, you set this value to a COM port that does not exist for the system on which your application is run, an error will be generated.

The Settings property allows you to specify the baud rate, parity, and the number of data bits and stop bits. By default, the baud rate is set at 9600. The parity setting is for data validation. It is commonly not used, and set to "N". The data bits setting specifies the number of bits that represent a chunk of data. The stop bit indicates when a chunk of data has been received.

Once youve specified which port to open and how data communication is to be handled, you use the PortOpen property to establish the connection. It is a Boolean value, True or False. If, however, the port is not functional, if the CommPort property is set incorrectly, or if the device does not support the settings youve specified, an error will be generated or the external device may not work correctly. Setting the value of the PortOpen property to False closes the port.

Working with a Modem

In most cases, you will use the Communications control to program your application to work with a modem. With the Communications control, you can use the standard Hayes-compatible command set to dial a phone number, or connect to and interact with another modem.

Once the serial port connection has been established using the CommPort, Settings, and PortOpen properties, you use the Output property to activate and interact with the modem. The Output property is used to issue commands which control the interaction between one modem and another. For example:

```
' Activate the modem and dial a phone number.
MSComm1.Output = "ATDT 555-5555" & vbCr
```

In the example above, the command "AT" initiates the connection, "D" dials the number, and "T" specifies touch tone (rather than pulse). A carriage return character (vbCr) must be specified when outputting to a terminal. You do not need to add the return character when outputting byte arrays.

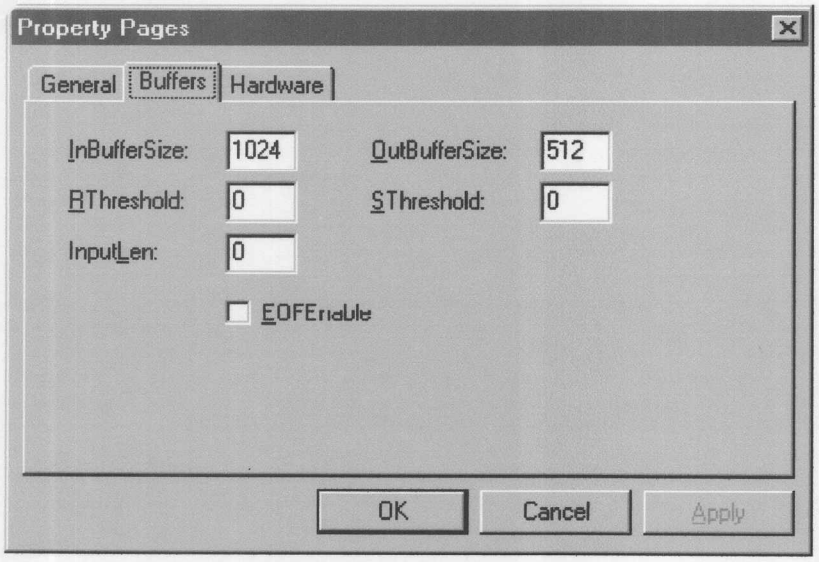
When a command is successfully processed, an "OK" result code will be returned. You can test for this result code to determine if a command was processed successfully.

For More Information For a complete list of Hayes-compatible commands, check your modem documentation.

Setting Receive and Transmit Buffer Properties at Design Time

When a port is opened, receive and transmit buffers are created. To manage these buffers, the Communications control provides you with a number of properties that can be set at design time using the controls Property Pages.

Setting buffer properties at design time



Buffer Memory Allocation

The InBufferSize and OutBufferSize properties specify how much memory is allocated to the receive and transmit buffers. Each are set by default to the values shown above. The larger you make the number, the less memory you have available to your application. If, however, your buffer is too small, you run the risk of overflowing the buffer unless you use handshaking.

Note Given the amount of memory available to most PCs at this time, buffer memory allocation is less crucial because you have more resources available. In other words, you can set the buffer values higher without affecting the performance of your application.

The RThreshold and SThreshold Properties

The RThreshold and SThreshold properties set or return the number of characters that are received into the receive and transmit buffers before the OnComm event is fired. The OnComm event is used to monitor and respond to changes in communications states. Setting the value for each property to zero (0) prevents the OnComm event from firing. Setting the value to something other than 0 (1, for instance) causes the OnComm event to be fired every time a single character is received into either buffer.

For More Information See "The OnComm Event and the CommEvent Property" in this topic for more information on these properties.

The InputLen and EOFEnable Properties

Setting the InputLen property to 0 causes the Communications control to read the entire contents of the receive buffer when the Input property is used. When reading data from a machine whose output is formatted in fixed-length blocks of data, the value of this property can be set appropriately.

The EOFEnable property is used to indicate when an End of File (EOF) character is found during data input. Setting this to True causes data input to stop and the OnComm event to fire to inform you that this condition has occurred.

For More Information See "Managing the Receive and Transmit Buffers" and "The OnComm Event and the CommEvent Property" in this topic for more information.

Managing the Receive and Transmit Buffers

As mentioned above, receive and transmit buffers are created whenever a port is opened. The receive and transmit buffers are used to store incoming data and to transmit outgoing data. The Communications control allows you to manage these buffers by providing you with a number of properties that are used to place and retrieve data, return the size of each buffer, and handle both text and binary data. Properly managing these buffers is an important part of using the Communications control.

The Receive Buffer

The Input property is used to store and retrieve data from the receive buffer. For example, if you wanted to retrieve data from the receive buffer and display it in a text box, you might use the following code:

```
txtDisplay.Text = MSComm1.Input
```

To retrieve the entire contents of the receive buffer, however, you must first set the InputLen property to 0. This can be done at design or run time.

You can also receive incoming data as either text or binary data by setting the InputMode property to one of the following Visual Basic constants: comInputModeText or comInputModeBinary. The data will either be retrieved as string or as binary data in a Byte array. Use comInputModeText for data that uses the ANSI character set and comInputModeBinary for all other data, such as data that has embedded control characters, Nulls, etc.

As each byte of data is received, it is moved into the receive buffer and the InBufferCount property is incremented by one. The InBufferCount property, then, can be used to retrieve the number of bytes in the receive buffer. You can also clear the receive buffer by setting the value of this property to 0.

The Transmit Buffer

The Output property is used to send commands and data to the transmit buffer.

Like the Input property, data can be transmitted as either text or binary data. The Output property, however, must transmit either text or binary data by specifying either a string or Byte array variant.

You can send commands, text strings, or Byte array data with the Output property. For example:

```
' Send an AT command
MSComm1.Output = "ATDT 555-5555"

' Send a text string
MsComm1.Output = "This is a text string"

' Send Byte array data
MSComm1.Output = Out
```

As previously mentioned, transmit lines must end with a carriage return character (vbCr). In the last example, `Out` is a variable defined as a Byte array: `Dim Out() As Byte`. If it were a string variant, it would be defined as: `Dim Out() As String`.

You can monitor the number of bytes in the transmit buffer by using the OutBufferCount property. You can clear the transmit buffer by setting this value to 0.

Handshaking

An integral part of managing the receive and transmit buffers is ensuring that the back-and-forth transmission of data is successful — that the speed at which the data is being received does not overflow the buffer limits, for example.

Handshaking refers to the internal communications protocol by which data is transferred from the hardware port to the receive buffer. When a character of data

arrives at the serial port, the communications device has to move it into the receive buffer so that your program can read it. A handshaking protocol ensures that data is not lost due to a buffer overrun, where data arrives at the port too quickly for the communications device to move the data into the receive buffer.

You set the Handshaking property to specify the handshaking protocol to be used by your application. By default, this value is set to none (comNone). You can, however, specify one of the other following protocols:

Setting	Value	Description
comNone	0	No handshaking (Default).
comXOnXOff	1	XOn/XOff handshaking.
comRTS	2	RTS/CTS (Request To Send/Clear To Send) handshaking.
comRTSXOnXOff	3	Both Request To Send and XOn/XOff handshaking.

The protocol that you choose depends upon the device to which youre connecting. Setting this value to comRTSXOnXOff supports both of the protocols.

In many cases, the communications protocol itself handles handshaking. Therefore, setting this property to something other than comNone may result in conflicts.

Note If you do set this value to either comRTS or comRTSXOnXOff, you need to set the RTSEnabled property to True. Otherwise, you will be able to connect and send, but not receive, data.

The OnComm Event and the CommEvent Property

Depending upon the scope and functionality of your application, you may need to monitor and respond to any number of events or errors which may occur during the connection to another device or in the receipt or transmission of data.

The OnComm event and the CommEvent property allow you to trap and check the value of communication events and errors.

When a communication event or error occurs, the OnComm event is fired and the value of the CommEvent property is changed. Therefore, if necessary, you can check the value of the CommEvent property each time the OnComm event is fired. Because communications (especially over telephone lines) can be unpredictable, trapping these events and errors allows you to respond to them appropriately.

The following table lists the communication events that will trigger the OnComm event. The values will then be written to the CommEvent property.

Constant	Value	Description
comEvSend	1	There are fewer than SThreshold number of characters in the transmit buffer.
comEvReceive	2	Received RThreshold number of characters. This event is generated continuously until you use the Input property to remove the data from the receive buffer.
comEvCTS	3	Change in Clear To Send line.
comEvDSR	4	Change in Data Set Ready line. This event is only fired when DSR changes from 1 to 0.
comEvCD	5	Change in Carrier Detect line.
comEvRing	6	Ring detected. Some UARTs (universal asynchronous receiver-transmitters) may not support this event.
comEvEOF	7	End Of File (ASCII character 26) character received.

The OnComm event is also triggered, and a value is written to the CommEvent property, when the following errors are encountered.

Setting	Value	Description
comEventBreak	1001	A Break signal was received.
comEventFrame	1004	Framing Error. The hardware detected a framing error.
comEventOverrun	1006	Port Overrun. A character was not read from the hardware before the next character arrived and was lost.
comEventRxOver	1008	Receive Buffer Overflow. There is no room in the receive buffer.

comEventRxParity	1009	Parity Error. The hardware detected a parity error.
comEventTxFull	1010	Transmit Buffer Full. The transmit buffer was full while trying to queue a character.
comEventDCB	1011	Unexpected error retrieving Device Control Block (DCB) for the port.

Visual Basic: MSComm Control

MSComm Control

See Also Example Properties **Methods** Events

The **MSComm** control provides serial communications for your application by allowing the transmission and reception of data through a serial port.

Syntax

MSComm

Remarks

The **MSComm** control provides the following two ways for handling communications:

- Event-driven communications is a very powerful method for handling serial port interactions. In many situations you want to be notified the moment an event takes place, such as when a character arrives or a change occurs in the Carrier Detect (CD) or Request To Send (RTS) lines. In such cases, use the **MSComm** control's **OnComm** event to trap and handle these communications events. The **OnComm** event also detects and handles communications errors. For a list of all possible events and communications errors, see the **CommEvent** property.

- You can also poll for events and errors by checking the value of the **CommEvent** property after each critical function of your program. This may be preferable if your application is small and self-contained. For example, if you are writing a simple phone dialer, it may not make sense to generate an event after receiving every character, because the only characters you plan to receive are the OK response from the modem.

Each **MSComm** control you use corresponds to one serial port. If you need to access more than one serial port in your application, you must use more than one **MSComm** control. The port address and interrupt address can be changed from the Windows Control Panel.

Although the **MSComm** control has many important properties, there are a few that you should be familiar with first.

Properties	Description
CommPort	Sets and returns the communications port number.
Settings	Sets and returns the baud rate, parity, data bits, and stop bits as a string.
PortOpen	Sets and returns the state of a communications port. Also opens and closes a port.
Input	Returns and removes characters from the receive buffer.
Output	Writes a string of characters to the transmit buffer.

MSComm Control Example

The following simple example shows basic serial communications using a modem:

```
Private Sub Form_Load ()
   ' Buffer to hold input string
   Dim Instring As String
   ' Use COM1.
   MSComm1.CommPort = 1
   ' 9600 baud, no parity, 8 data, and 1 stop bit.
   MSComm1.Settings = "9600,N,8,1"
   ' Tell the control to read entire buffer when Input
   ' is used.
   MSComm1.InputLen = 0
   ' Open the port.
   MSComm1.PortOpen = True
   ' Send the attention command to the modem.
   MSComm1.Output = "ATV1Q0" & Chr$(13) ' Ensure that
   ' the modem responds with "OK".
   ' Wait for data to come back to the serial port.
   Do
      DoEvents
   Buffer$ = Buffer$ & MSComm1.Input
   Loop Until InStr(Buffer$, "OK" & vbCRLF)
   ' Read the "OK" response data in the serial port.
   ' Close the serial port.
   MSComm1.PortOpen = False
End Sub
```

Note The **MSComm** control can use polling or an event-driven method to retrieve data from the port. This simple example uses the polling method. For an example of the event-driven method, see help for the OnComm event.

MSDN Home

MSComm Control Constants

See Also

Handshake Constants

Constant	Value	Description
comNone	0	No handshaking.
comXonXoff	1	XOn/XOff handshaking.
comRTS	2	Request-to-send/clear-to-send handshaking.
comRTSXOnXOff	3	Both request-to-send and XOn/XOff handshaking.

OnComm Constants

Constant	Value	Description
comEvSend	1	Send event.
comEvReceive	2	Receive event.
comEvCTS	3	Change in clear-to-send line.
comEvDSR	4	Change in data-set ready line.
comEvCD	5	Change in carrier detect line.
comEvRing	6	Ring detect.
comEvEOF	7	End of file.

rror Constants

Constant	Value	Description
comEventBreak	1001	Break signal received
comEventFrame	1004	Framing error
comEventOverrun	1006	Port overrun
comEventRxOver	1008	Receive buffer overflow
comEventRxParity	1009	Parity error
comEventTxFull	1010	Transmit buffer full
comEventDCB	1011	Unexpected error retrieving Device Control Block (DCB) for the port

InputMode Constants

Constant	Value	Description
comInputModeText	0	(Default) Data is retrieved through the **Input** property as text.
comInputModeBinary	1	Data is retrieved through the **Input** property as binary data.

Visual Basic: MSComm Control

OnComm Event

See Also Example Applies To

The **OnComm** event is generated whenever the value of the **CommEvent** property changes, indicating that either a communication event or an error occurred.

Syntax

Private Sub *object*_**OnComm ()**

The **OnComm** event syntax has these parts:

Part	Description
object	An object expression that evaluates to an object in the Applies To list.

Remarks

The **CommEvent** property contains the numeric code of the actual error or event that generated the **OnComm** event. Note that setting the **RThreshold** or **SThreshold** properties to 0 disables trapping for the **comEvReceive** and **comEvSend** events, respectively.

OnComm Event Example

The following example shows how to handle communications errors and events. You can insert code after each related Case statement, to handle a particular error or event.

```
Private Sub MSComm_OnComm ()
   Select Case MSComm1.CommEvent
   ' Handle each event or error by placing
   ' code below each case statement

   ' Errors
      Case comEventBreak    ' A Break was received.
      Case comEventFrame    ' Framing Error
      Case comEventOverrun    ' Data Lost.
      Case comEventRxOver    ' Receive buffer overflow.
      Case comEventRxParity    ' Parity Error.
      Case comEventTxFull    ' Transmit buffer full.
      Case comEventDCB    ' Unexpected error retrieving DCB]

   ' Events
      Case comEvCD    ' Change in the CD line.
      Case comEvCTS    ' Change in the CTS line.
      Case comEvDSR    ' Change in the DSR line.
      Case comEvRing    ' Change in the Ring Indicator.
      Case comEvReceive    ' Received RThreshold # of
                        ' chars.
      Case comEvSend    ' There are SThreshold number of
                       ' characters in the transmit
                       ' buffer.
      Case comEvEof    ' An EOF charater was found in
                       ' the input stream
   End Select
End Sub
```

CommEvent Property

See Also Example Applies To

Returns the most recent communication event or error. This property is not available at design time and is read-only at run time.

Syntax

object.**CommEvent**

The **CommEvent** property syntax has these parts:

Part	Description
object	An object expression that evaluates to an object in the Applies To list.

Remarks

Although the **OnComm** event is generated whenever a communication error or event occurs, the **CommEvent** property holds the numeric code for that error or event. To determine the actual error or event that caused the **OnComm** event, you must reference the **CommEvent** property.

The **CommEvent** property returns one of the following values for communication errors or events. These constants can also be found in the Object Library for this control.

Communication errors include the following settings:

Constant	Value	Description
comEventBreak	1001	A Break signal was received.
comEventFrame	1004	Framing Error. The hardware detected a framing error.
comEventOverrun	1006	Port Overrun. A character was not read from the hardware before the next character arrived and was lost.
comEventRxOver	1008	Receive Buffer Overflow. There is no room in the receive buffer.
comEventRxParity	1009	Parity Error. The hardware detected a parity error.
comEventTxFull	1010	Transmit Buffer Full. The transmit buffer was full while trying to queue a character.
comEventDCB	1011	Unexpected error retrieving Device Control Block (DCB) for the port.

Communications events include the following settings:

Constant	Value	Description
comEvSend	1	There are fewer than Sthreshold number of characters in the transmit buffer.
comEvReceive	2	Received Rthreshold number of characters. This event is generated continuously until you use the Input property to remove the data from the receive buffer.
comEvCTS	3	Change in Clear To Send line.
comEvDSR	4	Change in Data Set Ready line. This event is only fired when DSR changes from 1 to 0.
comEvCD	5	Change in Carrier Detect line.
comEvRing	6	Ring detected. Some UARTs (universal asynchronous receiver-transmitters) may not support this event.
comEvEOF	7	End Of File (ASCII character 26) character received.

Data Type

Integer

Error Messages (MS Comm Control)

See Also

The following table lists the trappable errors for the **MSComm** control.

Constant	Value	Description
comInvalidPropertyValue	380	Invalid property value
comSetNotSupported	383	Property is read-only
comGetNotSupported	394	Property is read-only
comPortOpen	8000	Operation not valid while the port is opened
	8001	Timeout value must be greater than zero
comPortInvalid	8002	Invalid Port Number
	8003	Property available only at run time
	8004	Property is read only at runtime
comPortAlreadyOpen	8005	Port already open
	8006	The device identifier is invalid or unsupported
	8007	The device's baud rate is unsupported
	8008	The specified byte size is invalid
	8009	The default parameters are in error
	8010	The hardware is not available (locked by another device)
	8011	The function cannot allocate the queues
comNoOpen	8012	The device is not open
	8013	The device is already open
	8014	Could not enable comm notification
comSetCommStateFailed	8015	Could not set comm state
	8016	Could not set comm event mask
comPortNotOpen	8018	Operation valid only when the port is open
	8019	Device busy
comReadError	8020	Error reading comm device
comDCBError	8021	Internal error retrieving device control block for the port